Technical Document 1652
September 1989

Using Broyden Updates to Approximate the Jacobian in a Semi-Implicit Backward Differentiation Code

L. D. Knight

NAVAL OCEAN SYSTEMS CENTER
San Diego, California 92152-5000

E. G. SCHWEIZER, CAPT, USN
Commander

R. M. HILLYER
Technical Director

ADMINISTRATIVE INFORMATION

This work was performed by Laura D. Knight, RESA Program Office, Code 4505, Naval Ocean Systems Center, for San Diego State University, as a partial fulfillment of the requirements for the degree of Master of Science in Applied Mathematics.

Released by
T. M. Fitzgerald, Acting Head
RESA Program Office

Under authority of
H. F. Wong, Head
Interoperability Division

FS

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE NOSC TD 1652	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Ocean Systems Center 6b. OFFICE SYMBOL (If applicable) NOSC	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) San Diego, California 92152-5000	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.: In-house; PROJECT NO.; TASK NO.; AGENCY ACCESSION NO.

11. TITLE (Include Security Classification)
USING BROYDEN UPDATES TO APPROXIMATE THE JACOBIAN IN A SEMI-IMPLICIT BACKWARD DIFFERENTIATION CODE

12. PERSONAL AUTHOR(S)
Laura D. Knight

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM TO	September 1989	29

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	ordinary differential equations (ODES) semi-implicit backward differentiation formula (SIBDF)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document describes the method used to solve a system of stiff ordinary differential equations (ODES).

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE PERSON L. D. Knight	22b. TELEPHONE (Include Area Code) 619-553-3969	22c. OFFICE SYMBOL Code 4505

DD FORM 1473, 84 MAR EDITION MAY BE USED UNTIL EXHAUSTED
UNCLASSIFIED

ABSTRACT

We wish to solve a system of stiff ordinary differential equations (ODEs), which are generally large and sparse. Solving a stiff system of ODEs requires the solution of a nonlinear equation involving the ODE Jacobian at each time step. This Jacobian is often formed by finite differences. Here we discuss Broyden's Method which will update the Jacobian as we go using information available in order to avoid the cost of recomputing the Jacobian. The update is a rank-one update.

This method is implemented in the code STRUT. STRUT is the original code STEP by Shampine and Gordon which implements an Adams predictor-corrector for solving nonstiff problems coupled with Stewart's implementation of a Semi-Implicit Backward Differentiation Formula (SIBDF) predictor-corrector for solving stiff problems.

Two models are discussed for implementing Broyden's Method over different time steps. The first model is based on updating the ODE Jacobian and the second model updates the Jacobian of the nonlinear equation which we will call the iteration matrix. Other considerations which are

discussed are convergence tests, when to perform an update, and how to determine when to form a new Jacobian and iteration matrix.

We use a Method of Lines semidiscretization of Burgers' equation and a reaction-diffusion model as our test problems. Results are given for each model and are compared with the original STRUT code, LSODE and LSODA.

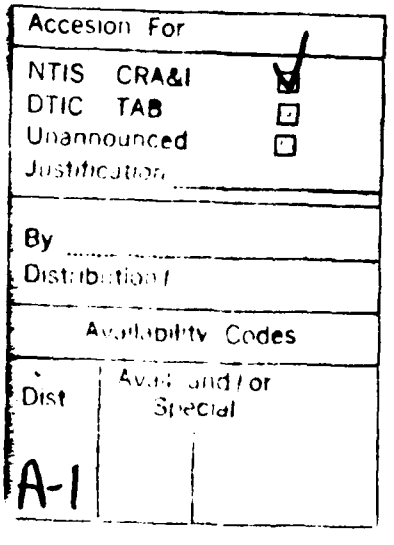

TABLE OF CONTENTS

LIST OF TABLES

Table

CHAPTER I

INTRODUCTION

We wish to solve the system of ordinary differential equations (ODEs)

$$y' = f(t, y) \quad y \in \Re^n \tag{1}$$

with initial condition

$$y(t_0) = y_0 .$$

The initial condition is the only information available and from this we must recover the solution values from what is known about the derivative. Let y_1 approximate the solution at t_1, i.e.,

$$y_1 \approx y(t_1) .$$

To find the solution a method is used which may be either a one-step method (using only current information) or a multi-step method (using past information as well).

Different techniques, often explicit in nature, are used to solve such systems. However, a problem referred to as stiffness may occur forcing one to use special techniques designed to handle this type of problem. We are using a code, STRUT [17], designed to detect stiffness and then switch to a method more suitable for solving stiff problems.

The problems being considered here are generally very large and sparse. The stiff solver requires the solution of a nonlinear system involving the ODE Jacobian at each time step. This Jacobian is often obtained by finite differences since analytic Jacobians are cumbersome for the user. The subject of this thesis is to use a technique by C. Broyden [3] to improve the Jacobian as we go using information from recent function evaluations and avoid the cost of recomputing the Jacobian. The software STRUT is modified to incorporate this method and is tested using two different reaction-diffusion partial differential equations. These PDEs are discretized in space using the Method of Lines [15] to yield a large system of stiff ODEs in time, making them suitable problems for testing in the ODE code STRUT.

CHAPTER II

STIFF ORDINARY DIFFERENTIAL EQUATIONS

Background

L.F. Shampine and C.W. Gear [13:2] define a stiff problem as one where "no solution component is unstable (no eigenvalue has a real part that is at all large and positive) and at least some component is very stable (at least one eigenvalue has a real part which is large and negative)." From the definition we see that a problem may be stiff on some intervals but not on others. This makes it important to detect the presence of stiffness, which will be discussed in the next section.

Stiffness occurs when the problem has a slowly varying solution. The majority of physical systems of interest will be stable, however systems characterized by very rapid decay to a slowly varying solution are the ones that may potentially be stiff [13]. Systems of equations which exhibit properties of stiffness are found in many different applications. Hall and Watt [6] and Shampine and Gear [13] provide several examples of stiff problems. These include problems in the areas of chemical reaction kinetics, guidance and control,

electrical transmission networks, heat and matter transfer, and nuclear reactions with species decaying at widely varying rates.

Often an explicit method such as Adams PECE or Runge Kutta will be used to solve a system of ODEs. A PECE algorithm is one which uses a predictor (P) followed by a derivative evaluation (E). A correction (C) is computed and the step is completed with another derivative evaluation (E). Explicit methods have finite stability regions. When solving a stiff problem, the fast component of the solution will decay out as the independent variable is advanced. When this occurs, the step size will be increased, eventually to a value outside the stability region. Using an unstable stepsize magnifies even small errors such as truncation error. This will be felt by the error estimator. The step must then be rejected, and the step size will be decreased to a value within the stability region. This phenomenon is known as ratcheting, and although the method will produce a solution, it will be very costly. The step size is being restricted by stability and not by accuracy. Shampine states in [12:442] that "stiffness means that the desired accuracy could be achieved with a step size much larger than that necesssary for a stable computation."

Mathematically, we can look at the problem of detecting stiffness through the Lipschitz constant [12]. In the system given by equation (1) we assume that f satisfies a Lipschitz condition such that

$$|f(t,u)-f(t,v)|\leq L|u-v| \qquad (2)$$

$$\forall (t,u),(t,v)\in \Re\times\Re^n$$

where $L>0$ is generally the smallest constant which satisfies the inequality (2). If the Lipschitz condition is satisfied, we know that equation (1) has a unique solution $y(t)$ in the neighborhood of the initial condition, and we only need to examine the behavior of f in a small neighborhood around $(t,y(t))$. If f has continuous first partial derivatives on this neighborhood, from the Mean Value Theorem we have

$$f(t,u)-f(t,v)=J(t,\xi)(u-v) \qquad (3)$$

where

$$J_{i,j}=\left[\frac{\partial f_i}{\partial y_j}(t,\xi)\right]$$

is the Jacobian and ξ is on the line connecting u and v.

The Lipschitz constant can be taken to be the maximum of the norm of the Jacobian on this neighborhood and we can approximate L by

$$L\approx|J(t,y(t))| \ .$$

Detecting Stiffness

When solving systems that are stiff, it is most efficient to use special software which is specifically designed to handle stiff problems. Many methods have effective techniques implemented which will detect a large Lipschitz constant, implying that the problem is either very stable or unstable. We expect that most problems being solved numerically are stable, which gives one reason to suspect that a problem is stiff if there is a large Lipschitz constant. Shampine [12] considers what additional evidence may be available to determine if a problem is stiff. This method was implemented by Stewart in the software STRUT [17], which is designed to use an Adams PECE until stiffness is detected and then switch to a method related to the Backward Differentiation Formula (BDF) while the problem is stiff.

Only a brief description of this technique is included here, with complete details in [12]. It is necessary to determine the step size which will ensure stability on the next time step if we use the current order. We let c_k be the radius of the largest disc contained within the stability region for the Adams method of order k. The values computed by Shampine are shown in Table 1. Let $\rho(J)$ be the spectral radius of the Jacobian, i.e., the maximum modulus of the eigenvalues of the Jacobian. If $|h|\rho(J) > c_k$, the computation may be unstable. This provides a means to determine by how

much the step size must be reduced to ensure stability. This step size is referred to as h_{stab}. Ideally one would compute $\rho(J)$, but this computation is expensive. We can, however, relate the Lipschitz constant to $\rho(J)$ as follows.

Table 1. Radius of the largest disc contained in the stability region.

	0	1	2	3	4	5	6	7	8	9	10
c_k	1.0	1.2	1.2	.93	.71	.52	.37	.26	.18	.12	.075

If $\frac{\partial f}{\partial y}$ exists, from equation (3) with $u = y_n$, the corrected solution, and $v = p_n$, the predicted solution at time n we have

$$\left\| \frac{\partial f}{\partial y} \right\| \sim \frac{\| f(t_n, y_n) - f(t_n, p_n) \|}{\| y_n - p_n \|} .$$

Since $\rho(J) \leq \left\| \frac{\partial f}{\partial y} \right\|$, we can use the above approximation to $\left\| \frac{\partial f}{\partial y} \right\|$ as our estimate of the Lipschitz constant which is available at no extra cost in the Adams PECE implementation.

The algorithm initially examines the Lipschitz constant to estimate the stepsize so that $h_{stab} L$ is within the stability region. If this h_{stab} is much smaller than the stepsize required by accuracy, then stiffness is suspected. When this occurs, a power iteration is performed using f to actually estimate the spectral radius of the ODE system. The power method is costly in f evaluations (usually seven

to ten function evaluations for convergence), and so is used as a last check to distinguish stiff problems from problems with large Lipschitz constants due to poor scaling or dominant complex eigenvalues. These problems are often more effectively solved by explicit methods. At the point when step size is being severely limited by these stability restrictions rather than by accuracy requirements, then the system is considered stiff and a switch is made to the BDF method.

Impact of Stiffness

An implicit method such as Adams-Moulton uses function iteration. Convergence of function iteration places a restriction on the stepsize that is similar to that imposed by stability. For other methods, such as the BDF, function iteration restricts h far more than stability. To make this clear we examine backward Euler (first order BDF).

We are to solve

$$y' = f(t, y) \quad y \in \Re^n .$$

With the backward Euler method,

$$y_n = y_{n-1} + h f(t_n, y_n)$$

is solved for y_n. We have a nonlinear equation to solve. Function iteration would take the form

$$y_n^{(i+1)} = y_{n-1} + h f(t_n, y_n^{(i)}) .$$

Convergence considerations restrict h by

$$\left\| y_n^{(i+1)} - y_n^{(i)} \right\| \le h \left| f(t_n, y_n^{(i)}) - f(t_n, y_n^{(i-1)} \right\| \le hL \left\| y_n^{(i)} - y_n^{(i-1)} \right\| .$$

For rapid convergence, this would require

$$|hL| << 1 .$$

However, in our stiff system, we have large L, which will put a severe restriction on the size of h. Now the problem is that convergence of fixed point for BDF is limiting the step size, whereas before stability of the Adams was the limiting factor, so we have gained nothing by using BDF instead of Adams. This indicates why function iteration is not a good technique for stiff problems.

The typical procedure for stiff problems is some variant of Newton's Method which approximates $f(t,y)$ locally by a linear system of equations, i.e.,

$$f(t,y) \approx f(t_n, y_n) + J(y - y_n)$$

and the iteration for backward Euler would be

$$y_n^{(i+1)} = y_{n-1} + h f(t_n, y_n^{(i)}) + hJ(y_n^{(i+1)} - y_n^{(i)})$$

where a reasonable approximation to J will usually provide convergence. If the differential equation is linear and J is the exact ODE Jacobian, the iteration converges in one step.

Due to the cost of solving the system at each iteration, it is necessary to provide a good initial approximation to $y_n^{(0)}$ such that only a few iterations are required. The

typical technique uses an explicit predictor followed by an iteration of the implicit corrector, which provides stability.

We see that stiffness dictates the use of a quasi-Newton method rather that the cheaper function iteration. From [2:297], quasi-Newton methods are referred to as "primarily those [methods] which are in some sense generalizations of the one-dimensional secant method." It would be costly to use a quasi-Newton method on a nonstiff problem when it is not required, which is why it is important to detect the presence of stiffness and then switch to a more effective method.

Semi-Implicit Backward Differentiation Formula (SIBDF)

Backward Differentiation Formulas (BDFs) are a popular technique being used in many general purpose codes. Semi-Implicit BDF (SIBDF) is the subject of Stewart's Ph.D. Dissertation [16] and has been implemented in STRUT [17]. The SIBDF differ from the BDF in that a fixed number of corrections are computed at each step. The material in this section is taken from Stewart's work [16].

The software STRUT implements both an Adams predictor-corrector (original code STEP by Shampine and Gordon [14]) and a SIBDF predictor-corrector for solving stiff systems. STRUT implements efficient techniques for handling the BDFs, which allow a variable step size on each

step. To demonstrate the method being used we will let the step size h, and the step number k, be constant. The BDF of order k asks that a polynomial passing through $k+1$ computed points have a derivative that matches the differential equation. STRUT uses a predictor-corrector implementation of the BDFs, where the predicted solution of order $k-1$, p_n, and the derivative p'_n, are obtained by extrapolating the polynomial passing through k past values of the computed solution. This polynomial is

$$\mathcal{P}_{k,n-1}(t) = y_{n-1} + \frac{(t-t_{n-1})}{h}\nabla y_{n-1} + \frac{(t-t_{n-1})(t-t_{n-2})}{h^2 2!}\nabla^2 y_{n-1} + \cdots$$

$$+ (t-t_{n-1})\cdots\frac{(t-t_{n-k+1})}{h^{k-1}(k-1)!}\nabla^{k-1} y_{n-1} \ .$$

The solution at the forward point is predicted by extrapolating the polynomial to yield

$$p_n = \mathcal{P}_{k,n-1}(t_n) = \sum_{r=0}^{k-1} \nabla^r y_{n-1}$$

and the predicted derivative

$$p'_n = \mathcal{P}'_{k,n-1}(t_n) = \frac{1}{h}\sum_{r=1}^{k-1} \delta_r \nabla^r y_{n-1}$$

where

$$\frac{\delta_r}{h} = \frac{d}{dt}\left[\frac{(t-t_{n-1})\cdots(t-s_{n-r})}{h^r r!}\right]\Bigg|_{t=t_n} = \frac{1}{h}\sum_{i=1}^{r}\frac{1}{i} \ .$$

We must now compute the correction. The implicit corrector will improve the solution using information from the differential equation. First, setting

$$y_n^{(0)} = p_n$$

m corrections are computed by solving the sequence of linear systems

$$\hat{G}(\hat{\alpha})\Delta y_n^{(i-1)} = -F(t_n, y_n^{(i-1)}) \tag{4}$$

$$y_n^{(i)} = y_n^{(i-1)} + \Delta y_n^{(i-1)} \qquad i = 1, \ldots, m \ .$$

The goal is to attempt to satisfy the method residual given by

$$F(t_n, y) = f(t_n, y) - p_n' - \frac{\delta_k}{h}(y - p_n) \ . \tag{5}$$

Since function iteration is not feasible as discussed earlier, a quasi-Newton method is formed by approximating the correction matrix $\hat{G}$, which is a parameterized approximation to a Newton matrix G,

$$G = \frac{\partial F}{\partial y} = \left. \frac{\partial f(t_n, y)}{\partial y} \right|_{y = y_n^{(i)}} - \frac{\delta_k}{h} I \ .$$

$\hat{G}$ is required to have the form

$$\hat{G}(\hat{\alpha}) = \hat{J} - \hat{\alpha} I$$

where

$$\hat{\alpha} \approx \frac{\delta_k}{h} \quad \text{and} \quad \hat{J} \approx J \ .$$

The linear system (4) is solved by doing a LU decomposition of $\hat{G}$ and using this matrix for all iterations until a decision is made to reevaluate $\hat{G}$. [An LU decomposition factors a matrix into two simple matrices; one is unit lower triangular

and the other is upper triangular.] We see that the evaluation of the Jacobian is an expensive part of the solution process, expecially since we wish to solve systems of equations which are large.

STRUT uses an Asymptotically ($h \to \infty$) Absolutely Stable method based on investigations by Klopfenstein [9] for controlling relative matrix error. Klopfenstein's work was expanded on by Krogh and Stewart [10]. As discussed in [10], this work allowed statements to be made which related predictor-corrector order to relative matrix error under the assumption of stability and allowed the conclusion that the predictor should be of order one lower than the corrector and the effective number of corrections should be $m = 2$.

In order to remain stable, the implementation of the SIBDF must control the eigenvalues of the relative error matrix

$$D = -\hat{G}^{-1}(G - \hat{G}) \ . \tag{6}$$

To do so, STRUT directly estimates the relative matrix error using knowledge of $\hat{\alpha}$, which is the current value of $\frac{\delta_k}{h}$, and an estimate of the eigenvalues of J of largest and smallest modulus [17]. Stewart shows that if two corrections are performed, a lower bound is available for the norm of

$$D^* = I - \hat{G}^{-1}G^* \ . \tag{7}$$

This lower bound is

$$\frac{\|\Delta y^{m+1}\|}{\|\Delta y^{m}\|} \leq |D^{*}| \approx |D| \ .$$

CHAPTER III

BROYDEN'S METHOD

Background

This chapter will first discuss the general theory behind Broyden's Method as applied to the nonlinear equation $F(y) = 0$. We will then discuss some applications of Broyden's Method which, like the general theory, do not discuss incorporating the method over different time steps in the process of solving an ODE. Last we will discuss the two models developed for the problem $F(t, y) = 0$, which is the problem that must be considered for practical and efficient implementation of Broyden's Method in STRUT.

Broyden's Method [3] is a technique which uses a secant approximation of the Jacobian of the nonlinear equation instead of an analytic or finite-difference Jacobian. Broyden's Method, named after C. Broyden, is often referred to as Broyden's update, because we are not approximating the Jacobian, but rather continuously updating an approximation of the Jacobian. This update is then implemented in other algorithms, such as quasi-Newton algorithms.

First, we will examine the system $F(y)=0$, which does not incorporate the dimension of time. A method would be used to generate a sequence of approximate solutions

$$y^{(i)} \quad , \quad i=1,2,\ldots$$

from an initial condition $y^{(0)}$, through an iterative process

$$y^{(i)} = y^{(i-1)} - (B^{(i-1)})^{-1} F(y^{(i-1)}) \ .$$

Here $B^{(i-1)}$ is the Jacobian matrix

$$\frac{\partial F}{\partial y}(y^{(i-1)})$$

or a Broyden approximation to the Jacobian.

For the technique, $B^{(i)}$ is obtained from $B^{(i-1)}$ by updating $B^{(i-1)}$ so that

$$B^{(i)} s^{(i)} = z^{(i)} \ ,$$

the secant equation, is satisfied, where

$$s^{(i)} = y^{(i)} - y^{(i-1)}$$

$$z^{(i)} = F(y^{(i)}) - F(y^{(i-1)}) \ .$$

This results in a rank one update of the form

$$B^{(i)} = B^{(i-1)} + \frac{(z^{(i)} - B^{(i-1)} s^{(i)})(s^{(i)})^T}{(s^{(i)})^T s^{(i)}} \ .$$

Reference [2] discusses implementation of Broyden's update in three quasi-Newton methods for solving stiff ODEs. The focus was on ODEs for which N was large and the Jacobian was sparse. The quasi-Newton methods examined were a

Doolittle LU updating procedure, and an implicit implementation of two updates discussed by Broyden. The methods were implemented in LSODE.

These methods, for a fixed value of t_n, all take a full quasi-Newton step

$$s^{(i)} = -(B^{(i-1)})^{-1} F(y^{(i-1)})$$

at each iteration of

$$y^{(i)} = y^{(i-1)} - (B^{(i-1)})^{-1} F(y^{(i-1)}) \ .$$

LSODE was modified to allow for occassional Broyden updates rather than updates at each iteration step. The potential use of Broyden updates over different time steps is not implemented or discussed in [2]. We would like to apply this update technique to a problem of the form $F(t, y) = 0$. This will require updating the Jacobian over different time steps. The methodology and notation used in the following is from Dennis and Schnabel [3].

Model A: Updating the ODE Jacobian

Our first model is

$$M_n(y) = f(p_n) + B_n(y - p_n) \tag{8}$$

which satifies

$$M_n(p_n) = f(p_n)$$

where

$$p_n = y_n^{(0)} \ .$$

For the Broyden update, we choose the requirement

$$M_n(p_{n-1}) = f(p_{n-1})$$

since we always have at least a PEC on each time step, and therefore $f(p)$ is always computed. This requirement yields

$$f(p_{n-1}) = f(p_n) + B_n(p_{n-1} - p_n)$$

and

$$B_n(p_{n-1} - p_n) = f(p_{n-1}) - f(p_n)$$

is the secant equation.

Let

$$s_n = p_{n-1} - p_n$$

$$z_n = f(p_{n-1}) - f(p_n)$$

then

$$B_n s_n = z_n \ . \tag{9}$$

When you extend the secant equation to n-dimensions equation (9) does not completely specify B_n when $n > 1$ since there are not enough conditions. For example, for $s_n \neq 0$, there exists an $n(n-1)$ dimensional subspace of matrices which satisfy equation (9). The quality of the secant approximation will therefore be a function of how well we choose from among the possibilities which satisfy equation (9).

The approach taken is to choose B_n by trying to minimize the change in the model (8) while still satisfying the secant equation (9). In [3] it is shown that the least change in the model which satisfies equation (9) is

$$B_n = B_{n-1} + \frac{(z_n - B_{n-1}s_n)s_n^T}{s_n^T s_n} .$$

Model B: Updating the Jacobian of the Nonlinear Equation

The second model investigates updating the Jacobian matrix of the nonlinear equation, also referred to as the iteration matrix. As discussed in the SIBDF section of Chapter II, the goal is to solve the residual equation (5) given as follows:

$$0 = F_n(y)$$

$$0 = f(y) - \mathcal{P}'_n(t_n)$$

$$0 = f(y) - \frac{d}{dt}\mathcal{P}_{n-1}(t_n) - \alpha_n(y - \mathcal{P}_{n-1}(t_n))$$

Here, α_n is the method coefficient at the current step, $\mathcal{P}_n(t)$ interpolates $y(t)$ at t_n and at past points, and $\mathcal{P}_{n-1}(t)$ is the polynomial from the previous step. From this the predicted solution and derivative are given by

$$p_n = \mathcal{P}_{n-1}(t_n)$$

$$p'_n = \mathcal{P}'_{n-1}(t_n) .$$

The secant update is derived as a least change update to a linearization of the nonlinear residual at each time step. Once again we choose to investigate a secant update which is valid from one time step to the next. Since it is possible to have a PEC iteration, we must develop a model which uses the predicted solution, $p_n = \mathcal{P}_{n-1}(t_n)$ and the derivative $f(p_n)$, since this is the only information available from the step t_n. The model investigated is presented below.

We first linearize $F_n(y)$ about p_n which yields our model at n

$$F_n^L(y) = F_n(p_n) + \frac{\partial F_n}{\partial y}(y - p_n)$$

$$= f(p_n) - p'_n + B_n(y - p_n)$$

where the superscript L denotes that this is a linearization. Now we consider the model from the previous time step t_{n-1},

$$F_{n-1}^L(y) = F_{n-1}(p_{n-1}) + \frac{\partial F_n}{\partial y}(y - p_{n-1})$$

$$= f(p_{n-1}) - p'_{n-1} + B_{n-1}(y - p_{n-1}) \ .$$

We then examine the difference of the two models which yields

$$F_n(y) - F_{n-1}(y) = f(p_n) - p'_n - [f(p_{n-1} - p'_{n-1}] \qquad (10)$$

$$- B_n(p_n - p_{n-1}) + (B_n - B_{n-1})(y - p_{n-1}) \ .$$

From this we choose the secant equation

$$F_n^L(p_n) - F_{n-1}^L(p_{n-1}) = B_n(p_n - p_{n-1}) \qquad (11)$$

where we are asking that B_n account for the change in the first residual of $F_n(y)$ and $F_{n-1}(y)$ with respect to the predictors p_n and p_{n-1} over a time step. When equation (11) is substituted into equation (10) this yields the model change

$$F_n^L(p_n) - F_{n-1}^L(p_{n-1}) = (p_n - B_{n-1})(y - p_{n-1}) \ .$$

Letting

$$s_n = p_n - p_{n-1}$$

$$z_n = F_n^L(p_n) - F_{n-1}^L(p_{n-1})$$

the rank one update to B_{n-1} which minimizes the change in the model is given by

$$B_n = B_{n-1} + \frac{(z_n - B_{n-1}s_n)s_n^T}{s_n^T s_n} \ .$$

CHAPTER IV

IMPLEMENTING BROYDEN'S METHOD IN STRUT

Convergence Tests

It is necessary to determine when to stop the iteration process in equation (5) at a given time step. The maximum number of iterations allowed is four. This requires a decision as to the value of m in $P(EC)^m$, with m possibly 2, 3, or 4. (The case of $m = 1$ will be considered seperately.) The convergence tests discussed here are similar to those implemented in LSODE [8]. However, our test for determining when to perform a Broyden update is order dependent, whereas the tests in the LSODE related investigations [2] were not.

The convergence test is based upon predicting the value of the next correction and determining if this correction is sufficiently small as to not significantly impact the solution. If this is the case, the iteration is terminated with no further corrections. The convergence test works as follows. On the first time step taken with the stiff method or after a new iteration matrix is formed, it is necessary to compute two corrections in order to estimate the convergence rate. The convergence rate is estimated by

$$crate = \frac{\|\Delta y^{(i)}\|}{\|\Delta y^{(i-1)}\|}$$

where $\Delta y^{(i)}$ was defined in equation (4). The test to determine when to terminate the iteration is

$$\|\Delta y^{i}\| \cdot crate < 0.05 \cdot eps \qquad (12)$$

where eps is the user specified error tolerance for the solution of the differential equations. If this condition is met the iteration is terminated and we proceed to the next time step. If the condition is not met, we proceed with another correction and the convergence rate is updated as follows for $i \geq 2$:

$$newrate = \frac{\|\Delta y^{i}\|}{\|\Delta y^{(i-1)}\|}$$

$$crate = \max(crate \cdot 0.2, newrate) \ .$$

Here, by choosing a maximum of the two rates, we are protecting against the possibility of a $crate$ which is near zero, since it is possible for $newrate$ to be arbitrarily small.

Allowing for the possibility of a PEC iteration can add additional efficiency to a method. The test is again based on predicting the value of the next correction. The first opportunity for a PEC occurs on the second time step taken with the stiff method or the second time step after a new matrix is formed. The $crate$ is saved from the first time step and the test for terminating the iteration is

$$\|\Delta y^{1}\| \cdot crate < 0.01 \cdot eps \ .$$

If this condition is satisfied we also set

$$crate = crate \cdot 1.5$$

to build conservatism into the test for the next time step. The next time step will again test for a potential PEC step, but will use a larger $crate$. The value of 1.5 is based on that used in LSODE. If the condition is not met we proceed with the iteration, and if it is met we take our BDF step.

Determining When to do a Broyden Update

Rather than doing Broyden updates at each time step, we would like to determine when it is necessary to perform an update. The update itself uses information from two time steps, so the first possible update occurs at time step two taken with the stiff method or after a new matrix is formed. The test is based on assessing the convergence rate relative to the tolerable relative matrix error ($|D|$ from equation (7)). This technique is based on earlier investigations by Klopfenstein [9] and an expansion of this by Krogh and Stewart [10] as was discussed in Chapter II. Stewart shows in [16]

$$crate \leq \|D^*\| \approx |D|$$

where D^* and D are defined (6) and (7).

From this work a table of values was generated which are dependent on both order and number of corrections. The test we use here for determining if a Broyden update should be done is

$$crate > K(m,k)$$

where k represents current order of the predictor and m is the number of corrections. In our test, $m=2$. Two corrections is our aim; if it appears that the convergence rate is not fast enough for $m=2$, we will do a Broyden update. Otherwise we proceed with the iteration without benefit of an update.

Determining When to Form a New Jacobian

It is necessary to determine when a new Jacobian is required. In other implementations of Broyden's Method [2], the decision to compute a new Jacobain is based on how many updates have been performed. Here we attempt to tie our decision to the acceptable relative matrix error.

The technique used is as follows. On the second and ensuing time steps, whenever a second correction is made and a Broyden update has not been done at this step, we test if

$$crate > K(2,k) .$$

If this is the case, it is determined that the relative matrix error is not tolerable. At this point an update will be made, and the iteration will be allowed to continue with potentially two additional corrections. If at this time the criteria of convergence in (12) is not met, appropriate flags will be set to form a new Jacobian and iteration matrix

on the next time step and we will proceed with a BDF step.

Further Comments on Implementation

Originally, Model A was developed to perform Broyden updates on each time step. The code was then modified to allow for updates on selected steps. This implementation used information from time steps n and $n-1$. The final version of Model A uses information from the current time step and from the time step when a Broyden update was last performed, i.e., n and some time $n-q$. Model A was implemented primarily to investigate how reliable Broyden's Method would be when based on predictor information over two different time steps. Using this model is expensive, since it requires that we refactor the iteration matrix after each Broyden update. This results in a cost of $O(n^2)$ for the update plus $O(n^3\backslash 3)$ for the refactorization.

The desired approach is to update the Jacobian of the nonlinear equation, which is Model B. The final approach used for Model B is to update this Jacobian using information from the current time step and from the last time step when a Broyden update was performed. The cost for this model is $O(n^2)$ for each Broyden update if QR factors were formed of the original iteration matrix. Hanson [7] discusses efficient techniques for implementing the update.

CHAPTER V

RESULTS AND CONCLUSIONS

Test Problem: Burgers' Equation

To examine the performance of Broyden's update as implemented in STRUT, Burgers' equation is used as a test problem [4]. Burgers' equation is

$$\frac{\partial u}{\partial t} - \mu \frac{\partial^2 u}{\partial x^2} + u \frac{\partial u}{\partial x} = 0 \qquad (13)$$

$$u(0,t) = u(1,t) = 0$$

$$u(x,0) = \sin \pi x \ .$$

Typically, u represents a position variable and μ is a viscosity parameter, the inverse of the Reynolds number. The equation models the balance between convection and diffusion. Burgers' equation is often used as a test problem, since exact solutions do exist for many choices of initial and boundary conditions.

In order to use Burgers' equation as a test problem, equation (13) is discretized in space using Method of Lines [15] to yield a time dependent system. The ODE system is

$$y'_1 = \mu \frac{-2.0y_1 + y_2}{\Delta x^2} - \frac{y_1 y_2}{2\Delta x}$$

$$y'_i = \mu \frac{y_{i-1} - 2.0y_i + y_{i+1}}{\Delta x^2} - y_i \frac{y_{i+1} - y_{i-1}}{2\Delta x} \qquad i = 2, n-1$$

$$y'_n = \mu \frac{y_{n-1} - 2.0y_n}{\Delta x^2} + \frac{y_n y_{n-1}}{2\Delta x}$$

where

$$\Delta x = \frac{1}{n+1}$$

$$y_i(t) \sim u(x_i, t)$$

and the initial conditions are

$$y_i = \sin\left(\frac{i\pi}{n+1}\right) .$$

Burgers' equation is also an appropriate one to use if examining stiffness. Given a value of μ, the number of points selected will determine whether or not the system is stiff. This allows many different combinations to be examined and results in an appropriate test set. As μ gets smaller the problem is less stiff; we focus on $\mu = 1$ and $\mu = .1$.

Test Problem: Reaction-Diffusion Model

The second test problem we consider is a reaction-diffusion model in one dimension [1]

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} + D(1 + \alpha - T)e^{-\frac{\delta}{T}} \quad , \quad t > 0 \quad , \quad 0 \le x < 1$$

$$T_x(0,t) = 0 \quad , \quad T(1,t) = 1 \quad , \quad t > 0$$

$$T(x,0) = 1 \quad , \quad 0 \le x \le 1$$

where

$$D = R \cdot \frac{e^{\delta}}{\alpha\delta} \ .$$

This models a single step reaction of a reacting mixture in a region from zero to one. This problem is tested with the values $\alpha = 1$, $\delta = 20$ and $R = 5$ where, u is the mass fraction of the reacting reactant, T is the reactant temperature, L is the Lewis number, α is the heat release, δ is the activation energy, D is the Damkohler number and R is the reaction rate ($R > 0.88$).

We again use Method of Lines to discretize this in space, with $n = 1/\Delta x$, yielding the ODE system

$$y'_1 = 2.0n^2(y_2 - y_1) + D(1.0 + \alpha - y_1)e^{-\frac{\delta}{y_1}}$$

$$y'_i = n^2(y_{i-1} - 2.0y_i + y_{i+1}) + D(1.0 + \alpha - y_i)e^{-\frac{\delta}{y_i}} \qquad i = 2, n-1$$

$$y'_n = n^2(y_{n-1} - 2.0y_n + 1.0) + D(1.0 + \alpha - y_n)e^{-\frac{\delta}{y_n}}$$

with initial conditions

$$y_i = 1.0 \qquad i = 1, n \ .$$

This problem is characterized by a sharp jump in temperature at $x = 0$, from near one to near $1 + \alpha$, which develops after approximately $t = .25$. A sharp flame front then forms and moves towards $x = 1$ at a rate proportional to $e^{\alpha\delta}/2(1 + \alpha)$, where it then reaches steady state.

Numerical Results

The two test problems above were used to examine the performance of the Broyden Method implemented in STRUT, which will now be referred to as LSTRT. The results are also shown for the codes STRUT [17], LSODE [8] and LSODA [11].

STRUT implements the SIBDF with one or two corrections at each time step. STRUT uses the code STEP by Shampine and Gordon [14] for nonstiff problems which is a $P_kEC_{k-1}E$ Adams implementation. STRUT incorporates stiffness detection based on estimates of the eigenvalues which allows automatic method selection between the Adams and SIBDF, which is used on stiff equations.

LSODE implements an Adams and BDF predictor-corrector, with predictor and corrector of the same order. Function iteration is used on the Adams corrector and a Newton-like iteration is used on the BDF corrector. A new Jacobian is formed with each new correction matrix. Automatic method selection is not incorporated and so the BDF was used to solve the problem for all time.

LSODA is the modification to LSODE which automatically selects Adams or BDF depending on problem behavior. Function iteration is still used on the Adams corrector, but two corrections are required to generate an estimate of the Lipschitz constant used for stiffness detection.

We would like to measure the amount of work done by each code to delivery an accuracy requested by the user. Here all problems were run with a required accuracy of 10^{-5}. "Work" is defined here as

1. Number of evaluations for the differential equation.
2. Number of evaluations of the ODE Jacobian.

The following measures were used for performance evaluation:

Error: The code LSODE was used with an accuracy request four orders of magnitude smaller than requested from the tested codes to compute "true solution" values at 15 geometrically spaced points in the integration. For each code, the absolute error is computed at these 15 points and the maximum value is printed.

Step: Number of steps taken.

NFE: Number of evaluations of the differential equation.

NG: Number of factorizations of the iteration matrix.

NJ: Number of evaluations of the ODE Jacobian.

K: Order of predictor used on the final step.

H: Step size used on the final step.

ENDTOL: If the code increased the user's requested error tolerance, then this value is printed, else blank.

NB: Number of Broyden updates performed.

NPEC: Number of PEC steps taken (LSTRT only).

The results are given for both Model A and Model B. Tables 2 through 5 show the numerical results for Model A, updating the ODE Jacobian. Tables 6 and 7 show the numerical results for Model B, updating the Jacobian of the nonlinear equation.

Table 2 shows a sample of problems when Model A is run using Burgers' Equation as a test problem. This table shows the impact of implementing the update on each time step. Although this is not the best way to implement the method, it does show a "best case" to use as a baseline for comparison. All three stiff problems shown in the table indicate that using the predictor information based on two different time steps is a feasible approach in implementing the update. In all three stiff problems, the number of function evaluations is comparable to that of STRUT, and as we would hope, no new Jacobian or iteration matrix is required with LSTRT. Also, since the updates are performed at each step, there are a significant number of PEC steps.

Table 3 shows the results of Model A when performing updates on each time step using the reaction-diffusion equation as the test problem. This problem is a more

difficult one to solve as we change the integration endpoint from .26 to .29. The stiff problems A4 and A5 use the endpoint .26 with two different grid sizes. Again these results are comparable to those of STRUT, with LSTKT actually requiring fewer function evaluations due to the update. We can see the difference between Burgers' equation and this problem by looking at the number of PEC steps performed. In this case, the percentage of PEC steps is significantly smaller than those seen in Table 2. Also, the number of Broyden updates exceed the number of steps taken, which means that there were some failed steps. (A Broyden update is performed before we determine if the step has failed.) In set A4, we see that LSODA did not switch to the stiff solver, and in fact in all the runs with the $npts = 8$, LSODA does not detect stiffness. Stiff problem A6 shows how much more difficult this problem is when run with a larger endpoint. Here all the codes work hard and require many function evaluations. For all three stiff problems, we see that there is a significantly greater error for all the codes when solving this problem.

Table 4 shows the results of Model A when performing updates on selected steps with Burgers' equation. We see that the number of Broyden updates required for all the stiff problems is small, but we now have fewer PEC steps and more function evaluations. We also begin to see some

degradation in error with LSTRT as is shown in stiff problems A9 and A10. However, we are again able to proceed with only one Jacobian and iteration matrix.

Table 5 shows the results for Model A updating on selected time steps using the reaction-diffusion equation as the test problem. We do significantly more Broyden updates for all the stiff problems, with a greater number required as the problem becomes more stiff (larger endpoint values). For the first time LSTRT must get a new Jacobian and iteration matrix as is shown in set A16. We see when running this problem how costly a solution can be, thus the savings that may be obtained by implementing a method such as Broyden's can be extremely desirable. However, as stated previously, Model A is not the preferred implementation because of the high cost of each update $(O(n^2)+O(n^3/3))$. However, we have shown that Broyden's Method based on models using the predictor information at different time steps can be implemented.

We would like to update the Jacobian of the nonlinear equation, i.e., Model B. We again examine the model where an update is performed on each time step. Unfortunately, it is rapidly apparent that this model is not the correct model to use. We would expect very good results with an update at each time step, which as shown in Table 6 is not the case. The results are shown for only one mildly stiff case, because we were unable to produce results for the very

stiff problems. This Table shows that we required four Jacobians and iteration matrices, which is more than STRUT or LSODA. The error is significantly worse and it uses more function evaluations than all three codes.

Table 7 shows the results for Model B when updates are performed on selected time steps. When we implement the update in this manner, we are able to run Burgers' equation with values of μ which result in stiff problems. The results are much better, which may indicate that our model is at least "close" to what we want.

A possible explanation as to why Model B is not the correct model is as follows. When we developed Model B we chose our secant equation to be

$$F_n^L(p_n) - F_{n-1}^L(p_{n-1}) = B_n(p_n - p_{n-1})$$

which paralleled the secant equation for $F(y) = 0$

$$F(y^i) - F(y^{i-1}) = B^i(y^i - y^{i-1})$$

where

$$B^i \approx \frac{\partial F}{\partial y} .$$

But, the algebra in obtaining Model B yields

$$g_n^L(p_n) - g_{n-1}^L(p_{n-1}) = f(p_n) - f(p_{n-1}) - p'_n + p'_{n-1}$$

where we would actually want

$$\begin{aligned} F_n^L(p_n) - F_{n-1}^L(p_{n-1}) &= f(p_n) - f(p_{n-1}) - \alpha_n(p_n - p_{n-1}) \\ &= (J^* - \alpha_n I)(p_n - p_{n-1}) . \end{aligned}$$

We have found through implementing Model B, that we are unable to deal with this changing method coefficient, α_n.

Conclusion

Our results from Model A, updating the ODE Jacobian, show that we can implement Broyden's Method based on models using the predicted information at different time steps. We would like to extend our work from updating the ODE Jacobian to updating the Jacobian of the nonlinear equation, our Model B. However, we find that the model selected is not sufficient, which indicates a need for a different model. Further investigation is required to develop such a model.

Table 2. Updating the ODE Jacobian at each time step (Burgers' Equation).

Problem No. A1: tolerance = .00001, μ = 1.0, npts = 7

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	1.4E-05	87	178	6	1	0	6.0E-01	1.0E-05		
LSTRT	2.3E-05	86	179	1	1	0	4.4E-01	1.0E-05	29	25
LSODA	1.4E-05	133	303	5	5	3	2.4E-01	1.0E-05		
LSODE	2.6E-05	69	169	12	12	4	2.5E-01	1.0E-05		

Problem No. A2: tolerance = .00001, μ = 1.0, npts = 15

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	2.5E-05	59	127	8	2	0	7.1E-01	1.0E-05		
LSTRT	4.6E-04	56	136	1	1	0	3.5E-01	1.0E-05	48	11
LSODA	2.5E-05	135	396	10	10	4	2.0E-02	1.0E-05		
LSODE	3.7E-05	69	264	12	12	4	2.7E-01	1.0E-05		

Problem No. A3: tolerance = .00001, μ = 1.0, npts = 31

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	4.8E-05	77	179	12	2	0	5.8E-01	1.0E-05		
LSTRT	1.1E-04	65	158	1	1	0	4.2E-01	1.0E-05	57	19
LSODA	4.8E-05	92	509	12	12	4	2.1E-01	1.0E-05		
LSODE	5.3E-05	69	456	12	12	4	2.6E-01	1.0E-05		

Table 3. Updating the ODE Jacobian at each time step (Reaction-Diffusion Problem).

Problem No. A4: tolerance = .00001, endpt = .26, npts = 8

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	1.0E-02	44	140	5	3	4	2.3E-04	1.0E-05		
LSTRT	1.3E-02	44	133	1	1	4	2.4E-04	1.0E-05	53	3
LSODA	1.0E-02	72	181	0	0	3	1.5E-04	1.0E-05		
LSODE	4.0E-02	68	237	15	15	3	1.4E-04	1.0E-05		

Problem No. A5: tolerance = .00001, endpt = .26, npts = 16

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	2.2E-02	48	180	10	4	4	3.1E-04	2.1E-05		
LSTRT	4.0E-02	51	150	1	1	4	2.6E-04	2.1E-05	59	9
LSODA	2.2E-02	72	284	9	9	4	2.5E-04	1.0E-05		
LSODE	6.4E-02	72	411	18	18	4	1.6E-04	1.0E-05		

Problem No. A6: tolerance = .00001, endpt = .27, npts = 16

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	1.6E-02	153	347	12	5	3	1.4E-04	2.1E-04		
LSTRT	2.6E-02	165	434	1	1	3	1.3E-04	2.1E-04	208	13
LSODA	1.6E-02	191	541	9	9	4	9.0E-05	1.0E-05		
LSODE	4.6E-02	198	822	33	33	5	1.3E-04	1.0E-05		

Table 4. Updating the ODE Jacobian on selected steps (Burgers' Equation).

Problem No. A7: tolerance = .00001, μ = 1.0, npts = 7

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	1.4E-05	87	178	6	1	0	6.0E-01	1.0E-05		
LSTRT	2.3E-05	86	221	1	1	0	2.9E-01	1.0E-05	5	2
LSODA	1.4E-05	133	303	5	5	3	2.4E-01	1.0E-05		
LSODE	2.6E-05	69	169	12	12	4	2.5E-01	1.0E-05		

Problem No. A8: tolerance = .00001, μ = 1.0, npts = 15

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	2.5E-05	59	127	8	2	0	7.1E-01	1.0E-05		
LSTRT	4.6E-05	56	161	1	1	0	5.6E-01	1.0E-05	10	3
LSODA	2.5E-05	135	396	10	10	4	2.0E-01	1.0E-05		
LSODE	3.7E-05	69	264	12	12	4	2.7E-01	1.0E-05		

Problem No. A9: tolerance = .00001, μ = 0.1, npts = 15

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	3.0E-05	80	183	3	1	3	1.3E-01	1.0E-05		
LSTRT	3.9E-04	79	200	1	1	3	1.3E-01	1.0E-05	3	2
LSODA	3.0E-05	101	251	3	3	4	9.3E-02	1.0E-05		
LSODE	4.6E-05	55	199	9	9	4	1.3E-01	1.0E-05		

Problem No. A10: tolerance = .00001, μ = 1.0, npts = 31

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	4.8E-05	77	179	12	2	0	5.8E-01	1.0E-05		
LSTRT	1.1E-04	65	202	1	1	0	5.4E-01	1.0E-05	15	4
LSODA	4.8E-05	92	509	12	12	4	2.1E-01	1.0E-05		
LSODE	5.3E-05	69	456	12	12	4	2.6E-01	1.0E-05		

Problem No. A11: tolerance = .00001, μ = 0.1, npts = 31

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	3.9E-05	42	171	5	3	4	1.3E-01	1.0E-05		
LSTRT	7.4E-05	42	146	1	1	4	1.4E-01	1.0E-05	9	0
LSODA	3.9E-05	87	369	7	7	5	1.5E-01	1.0E-05		
LSODE	6.6E-05	55	343	9	9	4	1.3E-01	1.0E-05		

Table 5. Updating the ODE Jacobian on selected time steps (Reaction-Diffusion Problem).

Problem No. A12: tolerance = .00001, endpt = .26, npts = 8

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	1.0E-02	44	140	5	3	4	2.3E-04	1.0E-05		
LSTRT	1.3E-02	45	170	1	1	4	2.5E-04	1.0E-05	12	0
LSODA	1.0E-02	72	181	0	0	3	1.5E-04	1.0E-05		
LSODE	4.0E-02	68	237	15	15	3	1.4E-04	1.0E-05		

Problem No. A13: tolerance = .00001, endpt = .26, npts = 16

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	2.2E-02	48	180	10	4	4	3.1E-04	2.1E-05		
LSTRT	4.0E-02	51	188	1	1	4	2.6E-04	2.1E-05	15	0
LSODA	2.2E-02	72	284	9	9	4	2.5E-04	1.0E-05		
LSODE	6.4E-02	72	411	18	18	4	1.6E-04	1.0E-05		

Problem No. A14: tolerance = .00001, endpt = .27, npts = 8

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	4.2E-03	153	309	10	5	3	2.4E-04	1.0E-05		
LSTRT	4.9E-03	138	442	1	1	3	2.7E-04	1.0E-05	35	2
LSODA	4.2E-03	195	453	0	0	3	2.2E-04	1.0E-05		
LSODE	1.6E-02	198	597	37	37	5	1.6E-04	1.0E-05		

Problem No. A15: tolerance = .00001, endpt = .27, npts = 16

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	1.6E-02	153	347	12	5	3	1.4E-04	2.1E-05		
LSTRT	2.6E-02	149	462	1	1	4	1.3E-04	2.1E-05	35	2
LSODA	1.6E-02	191	541	9	9	4	9.0E-05	1.0E-05		
LSODE	4.6E-02	198	822	33	33	5	1.3E-04	1.0E-05		

Problem No. A16: tolerance = .00001, endpt = .28, npts = 8

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	3.1E-02	178	395	28	9	1	2.4E-04	1.0E-04		
LSTRT	3.8E-03	143	473	2	2	2	4.1E-04	1.0E-04	33	0
LSODA	3.1E-02	216	496	0	0	2	1.1E-04	1.0E-04		
LSODE	2.0E-01	190	719	50	50	2	4.7E-04	1.0E-04		

Problem No. A17: tolerance = .00001, endpt = .28, npts = 16

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	1.9E-02	252	502	16	6	3	1.5E-04	2.1E-05		
LSTRT	3.1E-02	243	737	1	1	4	1.4E-04	2.1E-05	57	5
LSODA	1.9E-02	300	775	9	9	4	1.0E-04	1.0E-05		
LSODE	5.4E-02	296	1112	43	43	4	1.2E-04	1.0E-05		

Table 6. Updating the Jacobian of the nonlinear equation at each time step (Burgers' Equation).

Problem No. B1: tolerance = .00001, μ = 0.1, npts = 15

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	3.0E-05	80	183	3	1	3	1.3E-01	1.0E-05		
LSTRT	3.9E-04	80	259	4	4	3	1.2E-01	1.0E-05	14	0
LSODA	3.0E-05	101	251	3	3	4	9.3E-02	1.0E-05		
LSODE	4.6E-05	55	199	9	9	4	1.3E-01	1.0E-05		

Table 7. Updating the Jacobian of the nonlinear equation on selected time steps (Burgers' Equation).

Problem No. B2: tolerance = .00001, μ = 1.0, npts = 7

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	1.4E-05	87	178	6	1	0	6.0E-01	1.0E-05		
LSTRT	2.3E-05	86	235	3	3	0	3.0E-01	1.0E-05	9	2
LSODA	1.4E-05	133	303	5	5	3	2.4E-01	1.0E-05		
LSODE	2.6E-05	69	169	12	12	4	2.5E-01	1.0E-05		

Problem No. B3: tolerance = .00001, μ = 1.0, npts = 15

CODE	ERROR	STEP	NFE	NG	NJ	K	ENDH	ENDTOL	NB	NPEC
STRUT	2.5E-05	59	127	8	2	0	7.1E-01	1.0E-05		
LSTRT	4.6E-05	56	221	4	4	0	3.1E-01	1.0E-05	15	0
LSODA	2.5E-05	135	396	10	10	4	2.0E-01	1.0E-05		
LSODE	3.7E-05	69	264	12	12	4	2.7E-01	1.0E-05		

REFERENCES

[1] S. Adjeric, and J.E. Flaherty, A Moving Finite Element Method with Error Estimation and Refinement for One-Dimensional Time Dependent Partial Differential Equations, SIAM J. Numer. Anal. **23** (1986), 778-796.

[2] P.N. Brown, A.C. Hindmarsh, and H.F. Walker, Experiments With Quasi-Newton Methods in Solving Stiff ODE Systems, SIAM J. Sci. Stat Comput. **6** (1985).

[3] J.E. Dennis Jr., and R.B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice-Hall, Inc. (1983).

[4] C. Fletcher, Burgers' Equation: A Model for all Reasons, J.Noye Ed. Numerical Solutions of Partial Differential Equations, North-Holland Publishing Company (1982), 139-235.

[5] I. Gladwell, and D.K. Sayers, Computational Techniques for Ordinary Differential Equations, Academic Press (1980).

[6] G. Hall, and J.M Watt, Modern Numerical Methods for Ordinary Differential Equations, Oxford University Press (1976).

[7] R.J. Hanson, Notes on a Secant Method for Solving the BDF Equation for Stiff ODE's, Applied Dynamics International MS Date=880301.

[8] A.C. Hindmarsh, LSODE and LSODI, Two New Initial Value Ordinary Differential Equation Solvers, ACM/SIGNUM Newsletter **15** (1980).

[9] R.W. Klopfenstein, Numerical Differentiation Formulas for Stiff Systems of Ordinary Differential Equations, RCA Review **32** (1971), 447-462.

[10] F.T. Krogh, and K. Stewart, Asymptotic ($h \to \infty$) Absolute Stability for BDFs Applied to Stiff Differential Equations, ACM Trans. on Math. Softw. **10** (1984), 45-57.

[11] L.R. Petzold, Automatic Selection of Methods for Solving Stiff and Nonstiff Systems of Ordinary Differential Equations, SIAM J. Sci. Stat. Comput. **4** (1983), 136-148.

[12] L.F. Shampine, Lipschitz Constants and Robust ODE Codes, J.T. Oden. Ed. Computational Methods in Nonlinear Mechanics, North Holland Pub. Co. (1980), 427-449.

[13] L.F. Shampine, and C.W. Gear, A User's View of Solving Stiff Ordinary Differential Equations, SIAM Review **21**, (1979).

[14] L.F. Shampine, and M.K. Gordon, Computer Solutions of Ordinary Differential Equations: The Initial Value Problem, W. H. Freeman and Co. (1975).

[15] R. Sincovec, and N. Madsen, Software for Nonlinear Partial Differential Equations, ACM Trans. Math. Softw. **1** (1975), 232-260.

[16] K. Stewart, Semi-Implicit Backward Differentiation Formulas, Ph.D. Dissertation University of New Mexico, (1987).

[17] K. Stewart, and F.T. Krogh, Adams and Semi-Implicit BDF for Solving Stiff and Nonstiff ODEs, ACM Trans. on Math. Software, To be submitted Aug 1989.